Exhibit A

To The Declaration of Kris Kimbrough In Support Of Kelora's Opposition to Defendants' Claim Construction Brief and Motion for Summary Judgment of Invalidity and Noninfringement

```
//
//    CPC
//
#include <ctype.h>
#include <direct.h>

#include "NAVIGATE.h"

#include "c:\accsysx\include\db4.h"      // AccSys header file
#include "C:\c600\utils\db4_win.h"
#include "AMP_DB.h"
#include "helpid.h"

int FindCurrentID(WORD, struct FEATURES *, int);
void ChangeGroupsState(char, char, struct FEATURES *, int);
void ReadRecords(struct FEATURES *, int);
void UpdateControlStatus(HWND, int, struct FEATURES *, int);
void MergeRecordLists(void);

char CPCPath[64];
short xScreen, yScreen;                          // Screen size (X & Y)
HBRUSH hWhiteBrush, hBlackBrush;
long MatchingParts[200], Matching2[200];
int  List1Len, List2Len;
char szTempBuffer[100];
BOOL bHelpCalled = FALSE;
LPSTR lpHelpFile = "C:\\NAVIGATE\\NAVIGATE.HLP";
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR lpszCmdLine,
int nCmdShow)
{
 /*********************************************************************/
 /* HANDLE hInstance;       handle for this instance                  */
 /* HANDLE hPrevInstance;   handle for possible previous instances    */
 /* LPSTR  lpszCmdLine;     long pointer to exec command line         */
 /* int    nCmdShow;        Show code for main window display         */
 /*********************************************************************/

 MSG        msg;           // MSG structure to store your messages
 int        nRc;           // return value from Register Classes
 long       nWndunits;     // window units for size and location
 int        nWndx;         // the x axis multiplier
 int        nWndy;         // the y axis multiplier
 int        nWidth;        // the resulting width and height for this
 int        nHeight;       // window

 strcpy(szAppName, "NAVIGATE");
 hInst = hInstance;
 if(!hPrevInstance)
   {
    /* register window classes if first instance of application       */
    if ((nRc = nCwRegisterClasses()) == -1)
      {
       /* registering one of the windows failed                      */
       LoadString(hInst, IDS_ERR_REGISTER_CLASS, szString, sizeof(szString));
       MessageBox(NULL, szString, NULL, MB_ICONEXCLAMATION);
       return nRc;
      }
```

```
    }

      xScreen = GetSystemMetrics (SM_CXSCREEN);       // Assign Global
Variables
      yScreen = GetSystemMetrics (SM_CYSCREEN);

      // Create a device independant size and location
      nWndunits = GetDialogBaseUnits();
      nWndx = LOWORD(nWndunits);
      nWndy = HIWORD(nWndunits);
      nWidth = ((260 * nWndx) / 4);
      nHeight = ((125 * nWndy) / 8);

 /* create application's Main window                                    */
 hWndMain = CreateWindow(
                szAppName,               /* Window class name           */
                "AMP Navigator",    /* Window's title              */
                WS_CAPTION      |        /* Title and Min/Max           */
                WS_SYSMENU      |        /* Add system menu box         */
                WS_MINIMIZEBOX  |        /* Add minimize box            */
                WS_MAXIMIZEBOX  |        /* Add maximize box            */
                WS_THICKFRAME   |        /* thick sizeable frame        */
                WS_CLIPCHILDREN |         /* don't draw in child windows areas
*/
                WS_OVERLAPPED,
                CW_USEDEFAULT, CW_USEDEFAULT,        /* Use default X, Y
*/
                        nWidth, nHeight,         // Width, Height of window
                NULL,                    /* Parent window's handle      */
                NULL,                    /* Default to Class Menu       */
                hInst,                   /* Instance of window          */
                NULL);                   /* Create struct for WM_CREATE */


 if(hWndMain == NULL)
   {
    LoadString(hInst, IDS_ERR_CREATE_WINDOW, szString, sizeof(szString));
    MessageBox(NULL, szString, NULL, MB_ICONEXCLAMATION);
    return IDS_ERR_CREATE_WINDOW;
   }

 cwCenter(hWndMain);
 mode = d_READONLY|d_SINGLE;

 ShowWindow(hWndMain, nCmdShow);             /* display main window      */

 while(GetMessage(&msg, NULL, 0, 0))         /* Until WM_QUIT message    */
   {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
   }

 /* Do clean up before exiting from the application                     */
 CwUnRegisterClasses();
 return msg.wParam;
} /*  End of WinMain                                                     */
```

```
/*************************************************************************/
/*                                                                       */
/* Main Window Procedure                                                 */
/*                                                                       */
/* This procedure provides service routines for the Windows events       */
/* (messages) that Windows sends to the window, as well as the user      */
/* initiated events (messages) that are generated when the user selects  */
/* the action bar and pulldown menu controls or the corresponding        */
/* keyboard accelerators.                                                */
/*                                                                       */
/*************************************************************************/
LONG FAR PASCAL WndProc(HWND hWnd, WORD Message, WORD wParam, LONG lParam)
{
 HMENU      hMenu=0;            /* handle for the menu                 */
 HBITMAP    hBitmap=0;          /* handle for bitmaps                  */
 HDC        hDC;                /* handle for the display device       */
 PAINTSTRUCT ps;                /* holds PAINT information             */
 int        nRc=0;              /* return code                         */

 static HICON hIcon;
 static short xIcon, yIcon, xClient, yClient;
 short x, y;

      switch (Message)
      {
      case WM_COMMAND:
            switch (wParam)
            {
            case IDM_F_EXIT:
                  SendMessage(hWnd, WM_CLOSE, 0, 0L);
                  break;

            case IDM_H_ABOUT:
            {
                  FARPROC lpfnABOUTBOXMsgProc;

                  lpfnABOUTBOXMsgProc =
MakeProcInstance((FARPROC)ABOUTBOXMsgProc, hInst);
                  nRc = DialogBox(hInst, (LPSTR)"ABOUTBOX", hWnd,
lpfnABOUTBOXMsgProc);
                  FreeProcInstance(lpfnABOUTBOXMsgProc);
            }
            break;

            case IDM_L_CPC:
                 {
                  FARPROC lpfnCPC2MsgProc;

                  lpfnCPC2MsgProc = MakeProcInstance((FARPROC)CPC2MsgProc,
hInst);
                  nRc = DialogBox(hInst, MAKEINTRESOURCE(100), hWnd,
lpfnCPC2MsgProc);
                  FreeProcInstance(lpfnCPC2MsgProc);
                 }
                 break;
```

```
            default:
                return DefWindowProc(hWnd, Message, wParam, lParam);
           }
        break;        /* End of WM_COMMAND  */

case WM_CREATE:
  hIcon = LoadIcon(hInst, szAppName);
        xIcon = GetSystemMetrics(SM_CXICON);
        yIcon = GetSystemMetrics(SM_CYICON);

        hWhiteBrush = CreateSolidBrush(0x00FFFFFF);
        hBlackBrush = CreateSolidBrush(0x00000000);
        getcwd(CPCPath, sizeof(CPCPath)-5);
        nRc = strlen(CPCPath);
        if (CPCPath[nRc - 1] != '\\')
              strcat(CPCPath, "\\");
        strcat(CPCPath, "CPC");
    break;

case WM_MOVE:
     break;

  case WM_SIZE:
        xClient = LOWORD (lParam);
        yClient = HIWORD (lParam);
        break;

  case WM_PAINT:
     memset(&ps, 0x00, sizeof(PAINTSTRUCT));
     hDC = BeginPaint(hWnd, &ps);

     // Included in case the background is not a pure color
     SetBkMode(hDC, TRANSPARENT);

        for (y = 25; y < yClient; y += 2 * yIcon)
              for (x = 25; x < xClient; x += 2 * xIcon)
                    DrawIcon(hDC, x, y, hIcon);

     EndPaint(hWnd, &ps);
     break;

case WM_CLOSE:  /* close the window  */
     DestroyWindow(hWnd);
     if (hWnd == hWndMain)
       PostQuitMessage(0);  /* Quit the application  */
    break;

case WM_DESTROY:
        DeleteObject(hWhiteBrush);
        DeleteObject(hBlackBrush);
        if (bHelpCalled)
              WinHelp(hWnd, lpHelpFile, HELP_QUIT, NULL);
        break;


default:
     return DefWindowProc(hWnd, Message, wParam, lParam);
```

```
      }
   return 0L;
}     /* End of WndProc */

/*************************************************************************/
/*                                                                       */
/* Dialog Window Procedure                                               */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
BOOL FAR PASCAL ABOUTBOXMsgProc(HWND hWndDlg, WORD Message, WORD wParam, LONG
lParam)
{
 switch(Message)
   {
    case WM_INITDIALOG:
       cwCenter(hWndDlg);
       /* initialize working variables */
       break; /* End of WM_INITDIALOG */

    case WM_CLOSE:
       /* Closing the Dialog behaves the same as Cancel */
       PostMessage(hWndDlg, WM_COMMAND, IDCANCEL, 0L);
       break; /* End of WM_CLOSE    */

    case WM_COMMAND:
       switch(wParam)
         {
          case IDOK:
             EndDialog(hWndDlg, TRUE);
             break;
          case IDCANCEL:
             /* Ignore data values entered into the controls      */
             /* and dismiss the dialog window returning FALSE     */
             EndDialog(hWndDlg, FALSE);
             break;
         }
       break;    /* End of WM_COMMAND                        */

    default:
      return FALSE;
   }
 return TRUE;
} /* End of ABOUTBOXMsgProc */



/*************************************************************************/
/*                                                                       */
/* nCwRegisterClasses Function                                           */
/*                                                                       */
```

```
/* The following function registers all the classes of all the windows  */
/* associated with this application. The function returns an error code */
/* if unsuccessful, otherwise it returns 0.                             */
/*                                                                      */
/************************************************************************/
int nCwRegisterClasses(void)
{
 WNDCLASS   wndclass;    /* struct to define a window class             */
 memset(&wndclass, 0x00, sizeof(WNDCLASS));

 /* load WNDCLASS with window's characteristics                         */
 wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_BYTEALIGNWINDOW;
 wndclass.lpfnWndProc = WndProc;
 /* Extra storage for Class and Window objects                          */
 wndclass.cbClsExtra = 0;
 wndclass.cbWndExtra = 0;
 wndclass.hInstance = hInst;
 wndclass.hIcon = LoadIcon(hInst, "AMP_FIND");
 wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
 /* Create brush for erasing background                                 */
 wndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
 wndclass.lpszMenuName = szAppName;   /* Menu Name is App Name */
 wndclass.lpszClassName = szAppName; /* Class Name is App Name */
 if(!RegisterClass(&wndclass))
   return -1;


 return(0);
} /* End of nCwRegisterClasses                                          */


/************************************************************************/
/*  cwCenter Function                                                   */
/*                                                                      */
/*  centers a window based on the client area of its parent             */
/*                                                                      */
/************************************************************************/
void cwCenter(HWND hWnd)
{
     RECT rect;

    GetWindowRect(hWnd, &rect);           /* get dialog box dimensions */
    rect.bottom -= rect.top;              /* compute window height */
    rect.right    -= rect.left;                 /* compute window width */

    MoveWindow(hWnd, (xScreen/2) - (rect.right/2), (yScreen/2) -
(rect.bottom/2),
                              rect.right, rect.bottom, FALSE);
}


/************************************************************************/
/*  CwUnRegisterClasses Function                                        */
/*                                                                      */
/*  Deletes any refrences to windows resources created for this         */
/*  application, frees memory, deletes instance, handles and does       */
```

```
    /*  clean up prior to exiting the window                              */
    /*                                                                    */
    /********************************************************************/
    void CwUnRegisterClasses(void)
 5  {
     WNDCLASS   wndclass;    /* struct to define a window class          */
     memset(&wndclass, 0x00, sizeof(WNDCLASS));

     UnregisterClass(szAppName, hInst);
10  }    /* End of CwUnRegisterClasses                                   */



15  /********************************************************************/
    /*                                                                  */
    /* Dialog Window Procedure                                          */
    /*                                                                  */
    /* This procedure is associated with the dialog box that is included in */
20  /* the function name of the procedure. It provides the service routines */
    /* for the events (messages) that occur because the end user operates   */
    /* one of the dialog box's buttons, entry fields, or controls.      */
    /*                                                                  */
    /********************************************************************/
25  BOOL FAR PASCAL CPC2MsgProc(HWND hWndDlg, WORD Message, WORD wParam, LONG
    lParam)
    {

          POINT point;
30        int i, State, rc, *LenPtr;
          long RecNum, *RecPtr;
          DWORD Index;
         static WORD LastHelpID = HELPID_NO_HELP;

35  #define RADIO_BUTTON          1
    #define LISTBOX               2
    #define LISTBOX_TEXT          3
    #define LISTBOX_RADIO         4

40  #define SELECTABLE                  0
    #define UNSELECTABLE          1
    #define MARKED                      2
    #define SELECTED              3
    #define FORCED                      4
45
    #include "cpc.inc"

          switch(Message)
          {
50        case WM_INITDIALOG:
                DBFFileHandle[CPC_DBF] = (DBF)0;
                OpenStructDatabase(CPCDatabase);
                if (DBFFileHandle[CPC_DBF] == (DBF)0)
                SendMessage(hWndDlg, WM_CLOSE, 0, 0L);
55        // Quit dialog box
            else
```

```
            SendMessage(hWndDlg, WM_COMMAND, ID_CLEAR_ALL, 0L);         //
Initialize entire dialog box
            cwCenter(hWndDlg);
            break; // End of WM_INITDIALOG

      case WM_CLOSE:
            PostMessage(hWndDlg, WM_COMMAND, IDCANCEL, 0L);
            break;

      case WM_CTLCOLOR:
//                if (HIWORD(lParam) == CTLCOLOR_BTN || HIWORD(lParam) ==
CTLCOLOR_LISTBOX)
            if (HIWORD(lParam) == CTLCOLOR_BTN)
            {
                  i = FindCurrentID(GetWindowWord(LOWORD(lParam), GWW_ID),
&Features[0], (int)(sizeof Features / sizeof Features[0]));
                        if (i > 0 && Features[i].CurState == SELECTED)
                        {
                              SetTextColor((HDC)wParam, RGB(255, 255, 255));
      // Text Color White
                              SetBkColor((HDC)wParam, RGB(0, 0, 0));
      // Black Background color
                              SetBkMode( (HDC)wParam, TRANSPARENT);

//                  UnrealizeObject(hBlackBrush);
//                  point.x = point.y = 0;
//                  ClientToScreen(hWndDlg, &point);
//                  SetBrushOrg(wParam, point.x, point.y);

                              return (DWORD)hBlackBrush;
                        }
            }
//                else if (HIWORD(lParam) == CTLCOLOR_MSGBOX)
//                SetTextColor(wParam, 0x00000000);                     // Text
Color Black
//                return ((DWORD) hWhiteBrush);
                  return (DWORD) NULL;

      case WM_COMMAND:
            switch(wParam)
            {
            case ID_LISTBOX_ARRANGE: // List box
                  if (HIWORD(lParam) == LBN_DBLCLK)
                        SendMessage(hWndDlg, WM_COMMAND, IDOK, 0L);
            else if (HIWORD(lParam) == LBN_SELCHANGE)
                  {
                        Index = SendDlgItemMessage(hWndDlg, wParam,
LB_GETCURSEL, 0, 0L);
                        if (Index != LB_ERR)
                              SendDlgItemMessage(hWndDlg, wParam, LB_GETTEXT,
(WORD)Index, (LONG)(LPSTR)szTempBuffer);
                        else
                              break;
                        for (i=0; i < (sizeof Features / sizeof Features[0]);
i++)
                              if (Features[i].DialogID == wParam &&
```

```
                                    (strcmp(szTempBuffer, Features[i].ItemText)
    == 0) )
                              {
                                    LastHelpID = Features[i].HelpID;
 5                                  if (Features[i].CurState != SELECTED)
          // KWK
                                    {
                                    ChangeGroupsState(Features[i].DBFeatureNum,
    SELECTABLE,
10                                            &Features[0], (int)(sizeof
    Features / sizeof Features[0]));
                                    Features[i].CurState = MARKED;
                                    }
                                    break;
15                            }
                        }
                        break;

                case IDCANCEL:
20                      CloseStructDatabase(CPCDatabase);
                        EndDialog(hWndDlg, FALSE);
                        break;

                case IDOK:                          // *** Search!!! ***
25                      for (i=0; i < (sizeof Features / sizeof Features[0]); i++)
                        {
                              if (Features[i].CurState == MARKED)
                              {
                                    if (List1Len < 1)
30                                  {
                                          RecPtr = &MatchingParts[0];
                                          LenPtr = &List1Len;
                                    }
                                    else
35                                  {
                                          RecPtr = &Matching2[0];
                                          LenPtr = &List2Len;
                                          List2Len = 0;
                                    }
40

                                    rc = dXgetrno(IndexPtr[Features[i].DBFeatureNum],
    Features[i].Abbrev, &RecNum);
                                    while (RecNum != 0L)
45                                  {
                                          *RecPtr = RecNum;             /* Add Record
    # to list */
                                          RecPtr++;                           /* Move
    ptr to next location */
50                                        ++(*LenPtr);                        /*
    Increase the list length */
                                          RecNum =
    FindNextKey(IndexPtr[Features[i].DBFeatureNum], Features[i].Abbrev);
                                    }
55                                  if (List2Len > 0)                   /* More than
    1 feature marked! */
                                          MergeRecordLists();
```

```
                                Features[i].CurState = SELECTED;
                                ReadRecords(&Features[0], (int)(sizeof Features /
    sizeof Features[0]));
                                UpdateControlStatus(hWndDlg, List1Len,
 5  &Features[0], (int)(sizeof Features / sizeof Features[0]));
                                InvalidateRect(GetDlgItem(hWndDlg,
    Features[i].DialogID), NULL, TRUE);
                                }
                          }
10                        break;

                case ID_HELP:
                          WinHelp(hWndDlg, lpHelpFile, HELP_CONTEXT,
    (DWORD)LastHelpID);
15                        bHelpCalled = TRUE;
                          break;

                case ID_CLEAR_ALL:
                          List1Len = List2Len = 0;                    /* Reset to 0
20  matching parts */
                          SetDlgItemText(hWndDlg, ID_QTY_MATCHING, (LPSTR)NULL);
                          SetDlgItemText(hWndDlg, ID_PART_NUMBER, (LPSTR)NULL);
                          for (i=0; i < (sizeof Features / sizeof Features[0]); i++)
                          {
25                              Features[i].CurState = SELECTABLE;
                                Features[i].Qty = 0;

                                switch (Features[i].IDType)
                                {
30                              case RADIO_BUTTON:
                                      EnableWindow(GetDlgItem(hWndDlg,
    Features[i].DialogID), 1); /* Enable */
                                      SendDlgItemMessage(hWndDlg, Features[i].DialogID,
    BM_SETCHECK, (WORD)0, 0L);    /* Un-mark it */
35                                    InvalidateRect(GetDlgItem(hWndDlg,
    Features[i].DialogID), NULL, TRUE);
                                      break;

                                case LISTBOX:
40                                    EnableWindow(GetDlgItem(hWndDlg,
    Features[i].DialogID), 1); /* Enable */
                                      SendDlgItemMessage(hWndDlg, Features[i].DialogID,
    LB_RESETCONTENT, 0, 0L);      /* Nothing selected */
                                      SendDlgItemMessage(hWndDlg, Features[i].DialogID,
45  LB_SETCURSEL, -1, 0L);  /* Nothing selected */
                                      break;

                                case LISTBOX_TEXT:
                                      SendDlgItemMessage(hWndDlg, Features[i].DialogID,
50  LB_ADDSTRING, 0, (LONG)(LPSTR)Features[i].ItemText);
                                      break;
                                }
                          }           /* end of for */
                          break;
55
                default:          /* default for all radio buttons */
```

```
                    i = FindCurrentID(wParam, &Features[0], (int)(sizeof Features
/ sizeof Features[0]));
                    if (i < 0)
                          break;
 5
                    State = Features[i].CurState;             /* Save the current
state of button */

                    //
10                  // Change all buttons in this group to SELECTABLE state
                    // unless the user has already picked this item and said
SEARCH
                    if (Features[i].CurState != SELECTED)
                              ChangeGroupsState(Features[i].DBFeatureNum,
15 SELECTABLE,
                                        &Features[0], (int)(sizeof Features /
sizeof Features[0]));

                    /* The following if is used to make radio buttons behave
20 differently
                          than normal by toggling on/off when repeatedly selected
*/
                    if (State == MARKED)
                    {
25                        SendDlgItemMessage(hWndDlg, Features[i].DialogID,
BM_SETCHECK, (WORD)0, 0L);    /* Un-mark it */
                          Features[i].CurState = SELECTABLE;
                    }
                    else
30                  {
                          if (Features[i].CurState != SELECTED)     // If Not
already SELECTED (hard selection)
                                    Features[i].CurState = MARKED;            // Flag
this as Marked (soft selection)
35                  }
                    LastHelpID = Features[i].HelpID;
                    break;
               }
            break;    // End of WM_COMMAND
40
        default:
            return FALSE;
       }
     return TRUE;
45 } // End of CPC2MsgProc Dialog BOX !!!!!!


/*********************************************************************/
/*                                                                   */
50 /* MergeRecordLists()                                              */
/*                                                                   */
/* This function is responsible for creating a list of records which is */
/* the subset of common records from 2 lists.  This function takes each */
/* record from list 2 and checks it against list 1, if it is found then */
55 /* the record is left in list 2, if it is not in list 1 then the record */
/* number is forced to 0.  After all records are checked in list 2, the */
/* records which are non-zero in list 2 are used to create a new list 1.*/
```

```
/*************************************************************************/
void MergeRecordLists(void)
{
      int Record1Ptr, Record2Ptr;
      BOOL Found;
      long *RecPtr;

      for (Record2Ptr=0; Record2Ptr < List2Len; Record2Ptr++)
      {
            Found = FALSE;
            for (Record1Ptr=0; Record1Ptr < List1Len; Record1Ptr++)
            {
                  if (MatchingParts[Record1Ptr] == Matching2[Record2Ptr])
                  {
                        Found = TRUE;
                        break;
                  }
            }
            if (!Found)                                    // Not Found, so
mark it as deleted
            Matching2[Record2Ptr] = 0L;
      }

      List1Len = 0;
      RecPtr = &MatchingParts[0];
      for (Record2Ptr=0; Record2Ptr < List2Len; Record2Ptr++)
      {
            if (Matching2[Record2Ptr] > 0)
            {
                  *RecPtr = Matching2[Record2Ptr];    // Add Record # to list
                  RecPtr++;                                    // Move ptr
to next location
                  List1Len++;                                  // Increase
the list length
            }

      }
}


/*************************************************************************/
/*                                                                       */
/* UpdateControlStatus()                                                 */
/*                                                                       */
/* This function is responsible for updating all radio buttons and list */
/* boxes contained in the dialog box.  This function is called after     */
/* the matching Qty for each feature has been determined.  The update    */
/* mainly consists of the following rules:                               */
/*                                                                       */
/*   1)   If the Qty for a radio button is 0 then it is unselec-         */
/*        table and is grayed and the button is empty (not checked)      */
/*   2)   If the Qty for a radio button is equal to the TotalQty of      */
/*        matching parts then the button is disabled (grayed).  If it    */
/*        is matching in all parts because it is a feature selected by   */
/*        the user then it will be displayed in inverse when the message*/
/*        for WM_CTL_COLOR is processed.                                 */
```

```
/*   3)   If the Qty for a list box item is 0 then it is deleted from   */
/*         the scroll list.                                              */
/*   4)   If the Qty for a list box item is equal to the TotalQty of    */
/*         matching parts, then the list box is disabled.                */
/*                                                                       */
/*   In addition to these main steps the Qty of matching parts is       */
/*   displayed and if there is only 1 part then the matching part       */
/*   number is also displayed.                                          */
/*************************************************************************/
void UpdateControlStatus(HWND hWndDlg, int TotalQty, struct FEATURES *Info, int
Size)
{
      int i;
      DWORD Index;

      wsprintf(szTempBuffer, "%d", TotalQty);
      SetDlgItemText(hWndDlg, ID_QTY_MATCHING, (LPSTR)szTempBuffer);

      if (TotalQty == 1)
            SetDlgItemText(hWndDlg, ID_PART_NUMBER,
(LPSTR)GetFieldValue(CPC_DBF, FIELD_PART_NUM));


      for (i=0; i < Size; i++)
      {
            switch (Info[i].IDType)
            {
            case RADIO_BUTTON:
                  if (Info[i].Qty == 0)                           // Not valid
                  {
                        EnableWindow(GetDlgItem(hWndDlg, Info[i].DialogID), 0);
// Disable
                        SendDlgItemMessage(hWndDlg, Info[i].DialogID,
BM_SETCHECK, (WORD)0, 0L);    // UnMark it
                  }
                  else if (Info[i].Qty == TotalQty)         // Forced/Selected
                  {
                        if (Info[i].CurState != SELECTED)   // If Not SELECTED
(hard selection)
                              EnableWindow(GetDlgItem(hWndDlg,
Info[i].DialogID), 0); // Disable when FORCED
                        SendDlgItemMessage(hWndDlg, Info[i].DialogID,
BM_SETCHECK, (WORD)1, 0L);    // Mark it
                  }
                  break;

            case LISTBOX_TEXT:
                  Index = SendDlgItemMessage(hWndDlg, Info[i].DialogID,
LB_FINDSTRING, -1, (LONG)(LPSTR)Info[i].ItemText);
                  if (Index == LB_ERR)                            // Item not
in list box
                        break;
                  if (Info[i].Qty == 0)                           // Not valid
                        SendDlgItemMessage(hWndDlg, Info[i].DialogID,
LB_DELETESTRING, (WORD)Index, 0L);
//                else if (Info[i].Qty == TotalQty)         // Forced/Selected
```

```
    //                      EnableWindow(GetDlgItem(hWndDlg, Info[i].DialogID), 0);
    // Disable List Box
                        break;
                }
        }           // end of for
    }


    /*********************************************************************/
    /*                                                                   */
    /* ReadRecords()                                                     */
    /*                                                                   */
    /* This function is responsible for reading each record number contained*/
    /* in the "MatchingParts" record list (for a length of "List1Len").     */
    /* Prior to reading the records all features Qty is reset to 0.         */
    /* After each record is read each feature value specified by that part  */
    /* (in separate fields) is located in the local data structure and its  */
    /* Qty is incremented. This procedure is repeated for each record.      */
    /*********************************************************************/
    void ReadRecords(struct FEATURES *Info, int Size)
    {
          char *ptr;
          int i, FeatNum, RecordIndex;

          for (i=0; i < Size; i++)
                Info[i].Qty = 0;

          for (RecordIndex=0; RecordIndex < List1Len; RecordIndex++)
          {
                GetRecValues(CPC_DBF, MatchingParts[RecordIndex]);

                for (FeatNum = 1; FeatNum < 10; FeatNum++)
                {
                      wsprintf(szTempBuffer, "FEAT%d", FeatNum);
                      ptr = GetFieldValue(CPC_DBF, szTempBuffer);
                      if (isalnum(*ptr))
                      {
                            for (i=0; i < Size; i++)
                            {
                                  if (strcmp(Info[i].Abbrev, ptr) == 0)
                                  {
                                        Info[i].Qty++;
                                        break;
                                  }
                            }
                      }
                }
          }
    }


    /*********************************************************************/
    /*                                                                   */
    /* ChangeGroupsState                                                 */
    /*                                                                   */
    /* This function is called as the user clicks on radio buttons.  Its    */
```

```
/* purpose is to update all features in a certain group to a have a     */
/* specified NewState value (usually "SELECTABLE").  A group would be   */
/* for example all Contact Plating types.                               */
/************************************************************************/
void ChangeGroupsState(char GroupNum, char NewState, struct FEATURES *Info, int
Size)
{
      int i;

      for (i=0; i < Size; i++)
      {
            if (Info[i].DBFeatureNum == GroupNum)
                  Info[i].CurState = NewState;
      }
}

/************************************************************************/
/*                                                                      */
/* FindCurrentID                                                        */
/*                                                                      */
/* This function is called whenever we need to know which record in our */
/* memory based data structure contains the information for a specific  */
/* dialog ID.                                                           */
/************************************************************************/
int FindCurrentID(WORD SelectedID, struct FEATURES *Info, int Size)
{
      int i;

      for (i=0; i < Size; i++)
      {
            if (Info[i].DialogID == SelectedID)
                  return i;
      }
      return -1;
}
```

```
static struct FEATURES Features[] =
{
    /*
    DialogID                IDType DBFeatureNum  Abbrev    CurState
ItemText        Qty          HELP_ID

================================================================================
================================
    */
    ID_LISTBOX_ARRANGE,      LISTBOX,         3,            "X",
SELECTABLE,    NULL,              0,        HELPID_NO_HELP,

    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1104 ",  SELECTABLE,
"11-4",             0,        HELPID_CPC_ARR_11_4,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1108 ",  SELECTABLE,
"11-8",             0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1109 ",  SELECTABLE,
"11-9",             0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1307 ",  SELECTABLE,
"13-7",             0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1309 ",  SELECTABLE,
"13-9",             0,        HELPID_CPC_ARR_13_9,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1405 ",  SELECTABLE,
"14-5",             0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1407 ",  SELECTABLE,
"14-7",             0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1703 ",  SELECTABLE,
"17-3",             0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1709 ",  SELECTABLE,
"17-9",             0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1714 ",  SELECTABLE,
"17-14",            0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1716 ",  SELECTABLE,
"17-16",            0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "1728 ",  SELECTABLE,
"17-28",            0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2203 ",  SELECTABLE,
"22-3",             0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2214 ",  SELECTABLE,
"22-14",            0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2216 ",  SELECTABLE,
"22-16",            0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2319 ",  SELECTABLE,
"23-19",            0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2324 ",  SELECTABLE,
"23-24",            0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2228 ",  SELECTABLE,
"22-28",            0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2337 ",  SELECTABLE,
"23-37",            0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2357 ",  SELECTABLE,
"23-57",            0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2363 ",  SELECTABLE,
"23-63",            0,        HELPID_NO_HELP,
    ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2307 ",  SELECTABLE,
"23-7",             0,        HELPID_NO_HELP,
```

```
        ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2313M",  SELECTABLE,
    "23-13M (combo)",   0,        HELPID_NO_HELP,
        ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2316M",  SELECTABLE,
    "23-16M (combo)",   0,        HELPID_NO_HELP,
        ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2322M",  SELECTABLE,
    "23-22M (combo)",   0,        HELPID_NO_HELP,
        ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2807 ",  SELECTABLE,
    "28-7",             0,        HELPID_NO_HELP,
        ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2813M",  SELECTABLE,
    "28-13M (combo)",   0,        HELPID_NO_HELP,
        ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2816M",  SELECTABLE,
    "28-16M (combo)",   0,        HELPID_NO_HELP,
        ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2822M",  SELECTABLE,
    "28-22M (combo)",   0,        HELPID_NO_HELP,
        ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2824 ",  SELECTABLE,
    "28-24",            0,        HELPID_NO_HELP,
        ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2837 ",  SELECTABLE,
    "28-37",            0,        HELPID_NO_HELP,
        ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2857 ",  SELECTABLE,
    "28-57",            0,        HELPID_NO_HELP,
        ID_LISTBOX_ARRANGE,      LISTBOX_TEXT,  3,        "2863 ",  SELECTABLE,
    "28-63",            0,        HELPID_NO_HELP,


        ID_SQ_FLANG_RECEPT,      RADIO_BUTTON,  1,        "SQFL ",  SELECTABLE,
    NULL,               0,        HELPID_NO_HELP,
        ID_FREE_HANG_RECEPT,     RADIO_BUTTON,  1,        "FREE ",  SELECTABLE,
    NULL,               0,        HELPID_NO_HELP,
        ID_FEED_RECEPT,          RADIO_BUTTON,  1,        "FEED ",  SELECTABLE,
    NULL,               0,        HELPID_NO_HELP,
        ID_PLUG,                 RADIO_BUTTON,  1,        "PLUG ",  SELECTABLE,
    NULL,               0,        HELPID_NO_HELP,

        ID_SERIES1,        RADIO_BUTTON,  2,        "SER1 ",  SELECTABLE,    NULL,
    0,        HELPID_CPC_SERIES,
        ID_SERIES1_VDE,          RADIO_BUTTON,  2,        "SER1V",  SELECTABLE,
    NULL,               0,        HELPID_CPC_SERIES,
        ID_SERIES2,        RADIO_BUTTON,  2,        "SER2 ",  SELECTABLE,    NULL,
    0,        HELPID_CPC_SERIES,
        ID_SERIES3,        RADIO_BUTTON,  2,        "SER3 ",  SELECTABLE,    NULL,
    0,        HELPID_CPC_SERIES,
        ID_SERIES4,        RADIO_BUTTON,  2,        "SER4 ",  SELECTABLE,    NULL,
    0,        HELPID_CPC_SERIES,

        ID_NO_CONTACTS,          RADIO_BUTTON,  4,        "NOCON",  SELECTABLE,
    NULL,               0,        HELPID_NO_HELP,
        ID_SQ_POSTED,      RADIO_BUTTON,  4,        "SQ   ",  SELECTABLE,    NULL,
    0,        HELPID_NO_HELP,
        ID_ROUND_POSTED,         RADIO_BUTTON,  4,        "ROUND",  SELECTABLE,
    NULL,               0,        HELPID_NO_HELP,
        ID_ROUND_SOLID,          RADIO_BUTTON,  4,        "FEEDP",  SELECTABLE,
    NULL,               0,        HELPID_NO_HELP,
        ID_SOLDER,               RADIO_BUTTON,  4,        "SOLDR",  SELECTABLE,
    NULL,               0,        HELPID_NO_HELP,

        ID_GOLD_SELECT,          RADIO_BUTTON,  5,        "GOLDS",  SELECTABLE,
    NULL,               0,        HELPID_NO_HELP,
```

```
    ID_GOLD,                RADIO_BUTTON,  5,        "GOLD ",  SELECTABLE,
NULL,               0,        HELPID_NO_HELP,
    ID_TIN,             RADIO_BUTTON,  5,        "TIN  ",  SELECTABLE,    NULL,
0,        HELPID_NO_HELP,
    ID_PLASTIC,         RADIO_BUTTON,  6,        "PL   ",  SELECTABLE,    NULL,
0,        HELPID_NO_HELP,
    ID_METAL,               RADIO_BUTTON,  6,        "MET  ",  SELECTABLE,
NULL,               0,        HELPID_NO_HELP,
    ID_SEX_NORMAL,          RADIO_BUTTON,  7,        "STDS ",  SELECTABLE,
NULL,               0,        HELPID_NO_HELP,
    ID_SEX_REVERSE,         RADIO_BUTTON,  7,        "REVS ",  SELECTABLE,
NULL,               0,        HELPID_NO_HELP,
    ID_TETRASEAL,       RADIO_BUTTON,  8,        "TETR ",  SELECTABLE,    NULL,
0,        HELPID_NO_HELP,
    ID_NO_TETRASEAL,        RADIO_BUTTON,  8,        "NOTET",  SELECTABLE,
NULL,               0,        HELPID_NO_HELP,
    ID_NUM_STANDARD,        RADIO_BUTTON,  9,        "STDN ",  SELECTABLE,
NULL,               0,        HELPID_NO_HELP,
    ID_NUM_REVERSE,         RADIO_BUTTON,  9,        "REVN ",  SELECTABLE,
NULL,               0,        HELPID_NO_HELP,
};
```

300773286.1